Referencje

dr inż. Ireneusz Szcześniak

jesień 2017

1 Niestała l-referencja

```
#include <cassert>
#include <iostream>

int
main()
{
  int x = 1;

  // BŁĄD: brak inicjalizacji.
  // int &a;

  // Stała c.
  const int c = 300000000;
  // BŁĄD: non-const l-reference cannot bind to a const.
  // int &r = c;

  // OK, z raz na zawsze zainicjalizowane.
  int &z = x, y = 2;

  // To samo, co wyżej, ale teraz wiadomo, do czego odnosi się &:
  // int y = 2, &z = x;

  // Ten zapis jest mylący, bo sugeruje, że & odnosi się też do y:
  // int& z = x, y = 2;

  // Wskaźnik na referencję to wskaźnik na obiekt.
  assert (&z == &x);

  // To nie jest reinicjalizacja!  Przypisujemy wartość zmiennej y do
  // zmiennej x.
  z = y;

  // x = 2
  std::cout << "x␣=␣" << x << std::endl;

  // Inicjalizujemy l-referencję na podstawie l-wartości z.
  int &zz = z;
  // Inicjalizacja referencji zz wyżej ma ten sam efekt co, taka
  // inicjalizacja:
  int &zx = x;
}
```

2 Referencja stała (stała l-referencja)

```
#include <string>

using namespace std;

int
foo(const string &)
{
}

int
main()
{
  int x = 1;
  // The reference binds to lvalue.
  const int &l1 = x;
  // The reference binds to rvalue.
  const int &l2 = 1;

  string s;
  // The function parameter reference binds to lvalue.
  foo(s);
  // The function parameter reference binds to rvalue, which is the
  // temporary created with the constructor taking a "const char *".
  foo("Hello!");
}
```

3 R-referencja

```
#include <iostream>

int main()
{
  // BŁĄD: brak inicjalizacji.
  // int &&a;

  int i;

  // BŁĄD: r-referencja nie może wskazać l-wartości.
  // int &&z = i;

  int &l = i;
  // BŁĄD: r-referencja nie może być inicjalizowana l-wartością (którą
  // jest l-referencja).
  // int &&r = l;

  // OK: wskazujemy r-wartość.
  int &&x = 1;

  // BŁĄD: Inicjalizacja r-referencji na podstawie l-wartości (x ma
  // nazwę, więc jest l-wartością)
  // int &&z = x;
}
```

4 Referencje a kontenery i tablice

Referencji nie można przechowywać w kontenerach i tablicach:

```
#include<vector>

int main()
{
  int x, y;

  // Kontenery mogą przechowywać wskaźniki.
  std::vector<int *> v = {&x, &y};
  // Ale referencji już nie.
  // std::vector<int &> v;

  // Tablice mogą przechowywać wskaźniki.
  int *a[] = {&x, &y};
  // Ale referencji już nie.
  // int &r[] = {x, y};
}
```

5 Referencje w parach i krotkach

Pary i krotki (quasi kontenery) mogą przechowywać referencje:

```
#include <iostream>
#include <utility>
#include <tuple>
#include <vector>

int main()
{
  int x = 1;

  std::pair<int &, int &> p(x, x);
  p.second = 2;

  std::cout << "x␣=␣" << x << std::endl;

  // BŁĄD: elementy pary muszą być zainicjalizowane.
  // std::pair<int &, int &> p2;

  std::tuple<int &, int &, int &> t(x, x, x);
  std::get<2>(t) = 3;

  std::cout << "x␣=␣" << x << std::endl;

  std::vector<std::pair<int &, int &> > v{p, p};
  v[1].second = 4;

  std::cout << "v[0].first␣=␣" << v[0].first << std::endl;
}
```

6 Wiązanie obiektu tymczasowego przez referencję

```
#include <iostream>
#include <string>

using namespace std;

struct A
{
  string m_name;

  A(const string &name): m_name(name)
  {
    cout << "ctor:␣" << m_name << endl;
  }

  ~A()
  {
    cout << "dtor:␣" << m_name << endl;
  }
};

int main()
{
  A a = A("a");

  {
    const A &r1 = a;
  }

  {
    const A &r2 = A("r2");

    {
      const A &R2 = r2;
    }

    cout << "Just␣checking." << endl;
  }

  {
    A &&r3 = A("r3");
  }
}
```